関数型プログラムとデータフローマシン (データ・セマンティクスの理論と実際に関する研究)

AUTHOR(S):

雨宮, 真人

CITATION:

雨宮, 真人. 関数型プログラムとデータフローマシン (データ・セマンティクスの理論と実際に関する研究). 数理解析研究所講究録 1982, 461: 79-101

ISSUE DATE:

1982-06

URL:

http://hdl.handle.net/2433/103138

数理解析研究所講究録
461巻 1982年 79-101

関数型プログラムとデータフローマシン

雨宮真人
（日本電信電話公社 武蔵野電気通信研究所）

1. はじめに

現在のハードウェア技術の進歩には目覚ましいものがあり，VLSIにみられるように高機能化，低コスト化の傾向を示している．一方ソフトウェアの低生産性の問題が顕在化しており，高度の処理機能が要求されるにつれて今後益々問題になるものと思われる．最近ではこれを背景として従来のノイマンの原理に依らない言語及び計算機アーキテクチャが，記述の平易性，処理の高速性の観点から注目されるようになってきた．

関数型プログラムは同時性の表現が容易，簡潔明瞭な記述能力などの点で，従来型プログラムにない優れた性質を持っている．これ迄，関数型言語の欠点として実現上の効率の悪さがあげられてきたが，近年のハードウェア技術の進歩を背景に再考する余地が出てきたようだ．

データフローマシンの方式は問題に内在する並列性をそのまま引出し，かつ関数型言語で書かれたプログラムを効率よ

く実行することのできるマシンアーキテクチャとして期待されている。

本稿では，まず従来型プログラムと関数型プログラムの特徴を考察し，関数型プログラムの持つ特質を議論する。次に関数型プログラムを実行するマシンとしてデータフローマシンを取り上げ，関数型プログラムとの親和性並び問題点について考察する。

2．従来型プログラムと関数型プログラム

従来型言語に共通する特徴は次の2点に集約される。

(1) 逐次的実行制御概念

(2) assignment（副作用）による計算

この2つの特徴は同時にまた従来型言語のもつ種々の問題の基本的要因ともなっている。

(1)の問題は強制的に実行順序が定められることである。プログラム中には互に関連のない独立な文が存在しており，これらは本来どの順序で（或いは並列に）実行してもかまわないが逐次的に実行され，並列実行の可能性を失っている。また独立な文もデータ依存関係にある文も，逐次的実行という観点から一次元的に順序づけられているためデータの依存関係を識別することが困難となり，プログラムの理解を著し

く低下させることになる。

(2) の問題は副作用によってプログラムのセマンティクスを不透明にするということである。assignment文によってメモリセルの内容が自由自在に書きかえられるため，変数とその値との対応は一意的でなく可読性が著しく低下することになる。またプログラムの正当性の検証が困難となり，ソフトウェア生産性の低下につながる。

一方 pure Lisp に代表される関数型言語[1][2][3][4]は，以下の特長を有している。

(1) プログラムを関数として捉える。

assignmentの概念はなく，変数はあくまで数学的変数を意味する。従来型言語の call by name のように引数として記憶セルの名前を受けとりそこへ値を書込むこと（副作用）により値を返すメカニズムは存在しない。

(2) 関数の値は引数値のみによって定まる。

assignment（副作用）がないので，関数の計算時に他の計算によって暗に影響を受けることはない。従って関数計算は互に独立となり並列処理の可能性が生じる。

(3) 関数の合成・結合が容易であり，引数として関数を受けとり，或は実行結果として関数を返す高階の関数を定義することができる。従って従来型言語に比べるとプログラムの

記述が簡単かつ明瞭になる。

上記の特徴をより詳しくみるため、図1に関数型言語 Valid[4] で書かれた、共通集合を求めるプログラムの例を示す。プログラムは逐次制御の概念である文の代わりに、式及び関数で表わされる。（x = f(…) の形をしたものを式と呼ぶ。）各変数は記憶セルではなく値そのものを示し、記号＝は右辺の計算結果に左辺の名前を定義することを表わす。（値名定義） clause … end はブロックを示し、この中での局所的な値名定義を可能にしている。式は；で区切られるが、データ依存関係に従って並列に評価される。

値には一意的な名前が付けられ、値を引数とする関数が実行されている間、値と名前の対応は不変である。

同一の部分式は同じ値を示すので、同一の値名と置換してもプログラムの意味は変わらない。このように置換による意味の不変性も関数型言語の特長の一つである。

図2に高階の関数 reduce を使って整数列の和・積を求めるプログラム例を示す。reduce は関数 add 或いは mul を受け取り、整数列 x に次々と add 或いは mul を適用していく。sum 及び product はこの reduce を使用して手短に書かれる。このように高階の関数を使用することにより、プログラムの記述が簡明になる。

```
intersection: function (u,v:list) return (list)
= if null(u) then nil
    else clause
          x = intersection(cdr(u),v);
          y = car(u);
          return (if member(y,v) then cons(y,x) else x)
         end;
```

図1. 共通集合を求めるValidプログラムの例

```
reduce: function (f:function,x:list,a:integer)
        return (integer)
= if null(x) then a
             else f(car(x),reduce(f,cdr(x),a));
add: function(x,y:integer) return (integer) = x+y;
mul: function(x,y:integer) return (integer) = x*y;
-- for example, sum and product are defined
-- using this as follows
( sum(x)     == reduce(add,x,0)  )
( product(x) == reduce(mul,x,1)  )
```

図2 高階関数の例

(4) リストやアレイのような構造データ全体が一つの値として扱かわれる。構造データ値を引数として受取り，また結果として返すことが出来る。但し一度作られた構造データは不変であり，変更を行なう場合はコピーが作られる。

以上，関数型言語の優れた性質について述べてきたが，関数型言語が従来型言語ほどの普通をみなかったのは，その実現上の効率の悪さ故であった。次節では，関数型言語がデー

タフローマシン上でどのように実行され，効率上の工夫がなされているかについて述べる．(5)(6)

3．データ駆動制御による関数型言語の実行

本節では特に並列処理性という観点から，データフローマシンと関数型言語の親和性について考察する．

データ駆動の原理では，演算はその必要なオペランドが全て揃ったときにいつでも実行可能となる．（図3）

データ駆動の実行原理は，従来型のマシンに比し次の様な特徴をもつ．

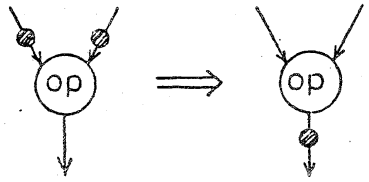


図3 データ駆動による実行

(1) オペランドが揃った演算は他の演算の状態とは無関係に実行できるので一度に複数の演算を実行することができる．（並列処理）

(2) 演算の実行は局所的であり，演算間のデータの授受はトークンとして明示的に行なわれるので他の演算に影響されることはない．（関数的処理）

(3) データ（トークン）により演算が駆動されるので実行順序を陽に制御する必要がなく，従って集中制御の必要性がない．（分散制御）

(4) データ保持のための記憶場所を演算とは独立に用意し

管理する必要がない。（記憶セル概念の排除）

これらの特長が関数処理にどのように生かされるか，以下考察する。

3．1　関数引数の並列評価

データ駆動の実行原理によると，関数の引数値が到着してから関数本体が実行されることになるので，引数の評価は内側から外側へと進む。（inner-most evaluation）　多レベルのリストを反転させるプログラム（図5）を例に，引数の並列評価効果をみてみよう。

ここで block 1 の部分を等価的に

```
fulrev(cdr(x),cons(fulrev(car(x),nil),y))
```

と書くことができる。（図5のデータフローグラフ参照）

関数 fulrev は2つの引数を持ち，各引数 cdr（x）と cons（…）は並列に評価される。さらに cons の2つの引数 fulrev（…）と y についても同様のことが繰返される。ソーステキスト上，引数のネストに制限があっても，実行の段階でネストが深まっていくので，みかけよりかなり高い並列性を得ることができる。

3．2　関数本体の部分的実行

データ駆動の実行原理は全てのオペランドが揃うことであった。しかしこれをそのまま関数の起動に適用すると，全て

```
fulrev: function (x,y) return (list)
= case
   null(x)⟶y;
   atom(x)⟶x;
   others  clause
block1 {   u=cdr(x);
           v=fulrev(car(x),nil);
           w=cons(v,y);
           return fulrev(u,w)
       end
end
```

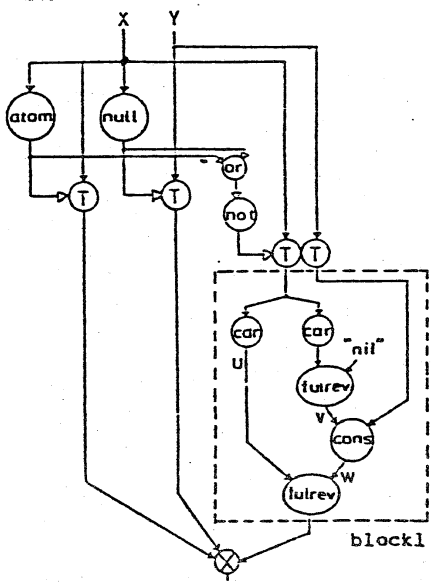


図4　関数fulrevとそのデータフローグラフ

の引数値が求まらない限り，関数本体の実行は開始されない。しかし引数の値が求まったものは直ちに関数本体へ送り込み，その値で実行できる所まで先に進めておけば，関数起動での待ち時間が省略でき実行効率を上げることができる。

関数起動　$[y_1, y_2, \cdots, y_n] = f(x_1, x_2, \cdots, x_m)$　のメカニズムを図5に示す。

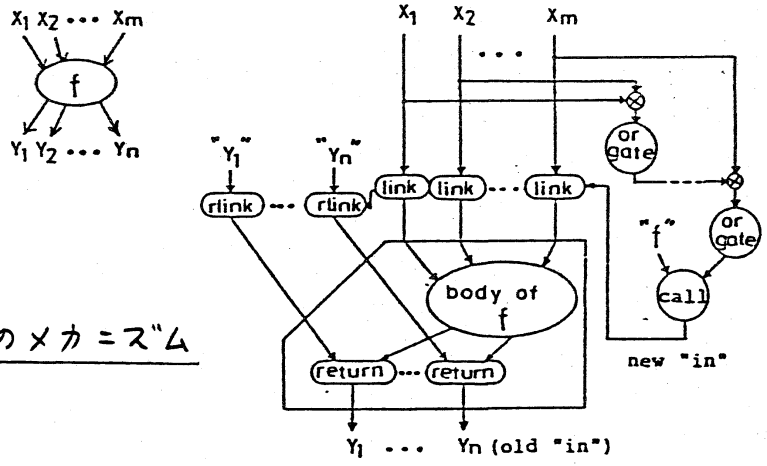


図5　関数起動のメカニズム

Orgateは先に到着したトークンだけを流す履歴を持ったノードである．Orgateの導入により関数起動ノードCallは引数トークン $x_1, x_2, \cdots x_m$ のいずれか一つが到着した時に発火する．ここでは関数本体は共有して使われるものとする．従ってトークンにはinstantiationを識別するためのinstance名（カラー）が付けられる．Callは関数本体が存在しなければ新たに関数本体を生成し，instance名を発生する．存在すれば新instance名を発生するだけである．

関数本体の起動準備が整うと，新instance名がlink, rlinkへ伝えられる．linkは引数トークンが到着する度に本体側へ引渡し，rlinkは戻り先情報（$y_1, y_2, \cdots, y_n$ 及び旧instance名）を本体中のreturnノードへ伝える．上記の起動メカニズムにより，関数本体の実行はトークンが到着する度に部分的に進行してゆく．また各関数値 $y_1, y_2, \cdots y_n$ もそれぞれのトークンが生成されると直ちに呼び側に返されるので，リンケージの効率を向上することができる．

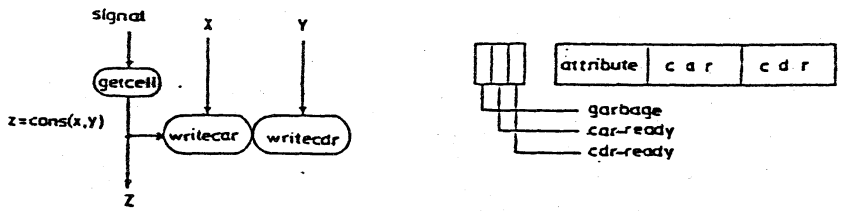
3.3 Lenient consとパイプライン処理

さて3.1で述べた機構は構造データの処理にもうまく機能するであろうか．リストを動的に操作していく問題の場合リストがconsまたはappendによって生成されている間，それを使用する関数の実行は待たされるため，部分的実行の効

果は薄れてしまう。しかしこの問題は予めセルを確保しておきセルへの書込みは値が求まった時に行なうLenient cons(7)の概念を導入することによって解決される。(8)

図6にLenient consの実現法を示す。cons(x, y)は3つの基本ノードgetcell, writecar, writecdrに分解される。データセルにはgarbageタグの他，値の到着を示すreadyタグが付されている。getcellはsignalトークンによって起動されると新セルを確保し，そのアドレスをcons結果を待つノード及びwritecar, writecdrに伝える。getcell時にはcar, cdrのreadyタグはオフにセットされ，そこへのアクセスは禁止されている。writecar, writecdrの各ノードはオペランドが到着したとき発火し，オペランドをcar部又はcdr部へ書き込み，readyタグをオンにしてアクセス禁止を解除する。

getcellの起動信号であるsignalトークンはこのcons



(a) Cons mechanism　　(b) Data cell structure

図6　Lenient consの実現法

を含む関数或いはブロックが開かれたときに生成される。

consのLeniencyにより，リストをパイプライン的に処理することが可能となる。例えばquick sortのプログラムは$\Theta(n)$の時間で計算することが出来る。(9) また，リストが全て作られる迄待つ必要がないので，関数が活性化されている時間が短くなり，資源の有効利用が図れる。

3.4 値名定義の効用

2で述べた通り，式の値には値名定義によって，一意的な名前がつけられる。これを利用して並列処理により効率を上げることが出来る。

同一の部分式が数箇所に現われる場合，その部分式を一度だけ計算し，残りの箇所にはそのコピーを送るようにすれば，何度も計算しなくてすむ。図1に示した共通集合を求めるプログラムの例ではcar(u)の計算結果にyという名前が付され，その値がmember(y, v)とcons(y, z)で参照される。ここで重要なことはintersection(cdr(u), v)とmember(y, v)の計算は並列に実行できるということである。この効果とLeniant consの効果とによりintersectionは$\Theta(m+n)$の時間で計算できる。(m, nは集合u, vの要素数である。) これを図7のように書くと，intersection(cdr(u), v)の計算はmember(car(u), v)の後で

か行なえないので計算時間は $O(m \times n)$ になってしまう。

```
intersection: function (u,v) return (list)
= if u=nil then nil else
  if member(car(u),v)
     then cons(car(u),intersection(cdr(u),v))
     else intersection(cdr(u),v);
```

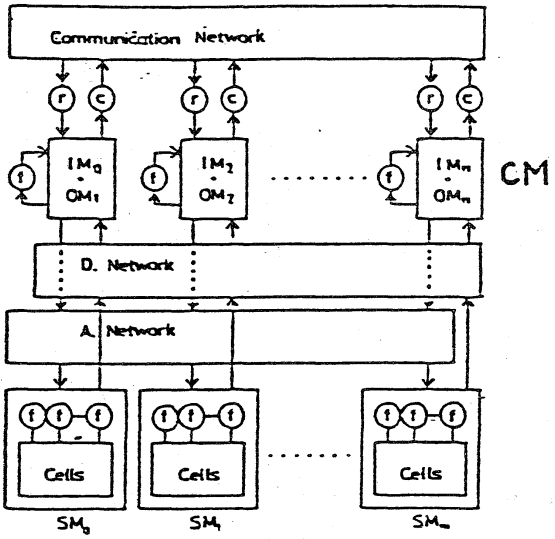
図7　値名定義を利用しないプログラム

4. データフローマシンのアーキテクチャ[5][6][10]

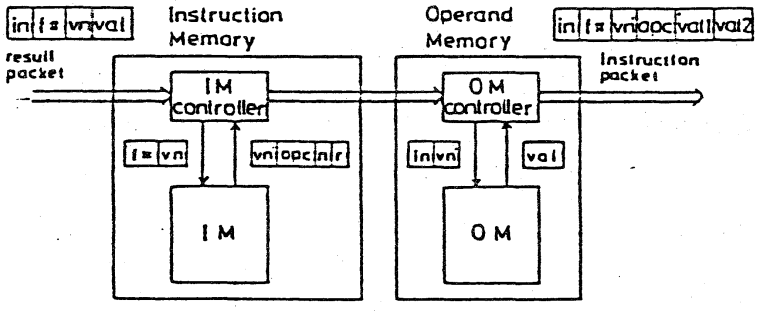
図8はリスト処理を指向したデータフローマシンの例である。ここでは色つきトークン方式を採用している。

本マシンは実行制御部である Control Module（CM），CM 間をつなぐ通信ネットワーク，複数のバンクから構成される構造メモリ（SM）及び CM-SM 間を結合する A-Network, D-Network からなる。（図7.(a)）

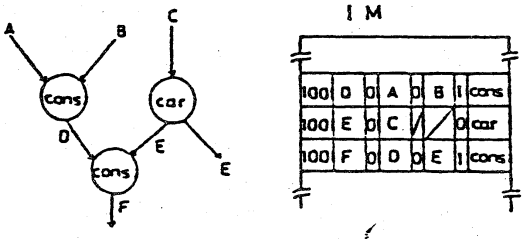
CM は図7（b）に示すように Instruction Memory（IM）と Operand Memory（OM），及び命令の取り出し，結果分配用 controller より構成される。IM は読出し専用の連想メモリで実現され プログラム（関数本体）が格納される。（図7(c)）　一方 OM は到着オペランドを保持するバッファとして使用される。図7（b）に於て，結果パケットが送られてくると，〈関数番号 f-number，バリュー名 vn〉を Key として IM が検索され，一致した（結果を待っている）命令が読出



(a) マシンの全体構成



(b) Instruction Memory と Operand Memory の構造



(c) データフロープログラム

図7. リスト処理向データフローマシン

される。一致した命令のオペランド数が1であれば，直ちに結果の値を基に命令パケットが作られる。オペランド数が2であれば，〈一致した命令のvalue名 vn，instance名 in〉

をkeyとしてOMが検索される。一致すれば該データを読出し，2オペランド型の命令パケットを作る。不一致なら，空きエントリーに結果パケットと第1／第2オペランドの区別を書込む。現在IMとOMの連想機能はハッシュ方式により実現している。

CM間の通信はプログラム実行時に生起する関数間のCall/Returnを契機として行なわれCall/Returnパラメタが CM間通信ネットワークを介して転送される。従ってCM間の論理的結合関係は木構造となる。CM間通信ネットワークはプログラムの構造及びCMの負荷に応じて，論理的に動的な木構造を実現する。(5)

SMは多バンク構成で，各バンクはさらに独立動作可能なcar, cdr, attribute, reference count, のフィールドに分割され，それぞれ専用の操作ユニットが設けられる。(10) CMから送出された構造体操作命令パケットはA-Networkでオペランドアドレスのデコードがなされ，対応するSMバンクへ転送される。D-NetworkはSMから送出された結果パケットを受取ると結果パケット中のIM番号を基に要求元CMへその結果パケットを送る。

このマシンの特徴はDennisのActivity Store(11)に相当する部分をIMとOMに分離し，メモリ使用効率を上げているこ

と，IMとOMの管理に連想機能を用いていること，リストの様な構造データの処理機構として，データの格納部と操作部を一体化しこれを多バンク構成することによりアクセスの負荷を分散させていること，等である．本マシンはCM-SM間の縦方向のパイプライン及びCM-CM間のパラレル実行により高度の並列処理を実現することを狙いとしている．

5．データフローマシンの問題点と解決策

データフローマシンを実用に供するには，まだ多くの問題が残されている．本節ではデータフローマシンの主要な問題点とその解決策について考察する．

5．1　データ駆動と要求駆動

データ駆動方式ではinnermostに引数評価が行なわれるので，不必要な引数でも全て評価されてしまう．例えば，

$f(p(x), g(x), h(x)) \equiv$ if $p(x)$ then $g(x)$ else $h(x)$

の計算で必要な引数は$p(x)$と$g(x)$, $h(x)$の内2つであるがinnermostに評価すると全ての引数が評価され場合によっては停止しなくなる．（条件実行に於るこの様な問題を解決するためデータ駆動ではスイッチノードを設けている．図4のデータフローグラフではこのスイッチノード㊦が用いられている．）また無限リストを作る関数

$$g(t) \equiv cons(x, g(t+1))$$

を用いて car(g(0)) の計算を innermost に行なうと，この計算は停止しない。

このように innermost 評価には case by case で処理したい場合でも無駄な計算を行ない，最悪の場合には停止しないことになる。

一方要求駆動方式では図8に示すように引数の値が必要になった時点で要求を出すので不要な計算を省くことができる。また要求駆動と lazy evaluation（cons(x, y) の評価で引数評価を行なわずセルだけを返す）の機能を組合せると，先程の無限リストの例にみられるような停止性の問題はなくなる。しかし要求が次々と最内側まで（値が得られる所まで）伝搬されねばならないので効率は悪い。例えば加減乗除を主体とする数値計算では，引数評価の段階ですぐに値が得られるため，要求を伝える為のオーバヘッドが深刻となり，この場合もデータ駆動の方が良い。

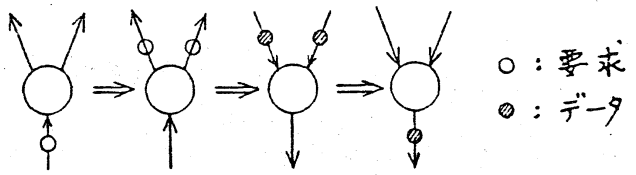


図8. 要求駆動の原理

記号処理などでは両方式の長所を融合するような方式が望まれるが，データ駆動に要求駆動的要素を取入れて lazy

evaluation の機能を持たせることは可能である。これは3.3節で述べた Lenient cons 機構を発展させた lazy cons により図9のように実現される。

Lazycons (f(x), g(y)) は getcell, wcarentry, wcdrentry, wcar, wcdr, 及び遅延評価のためのゲートに分解される。

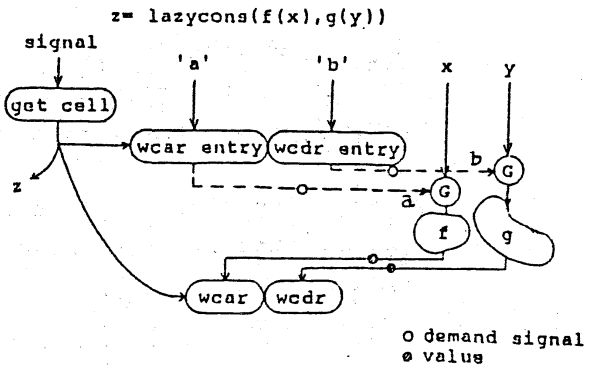


図9 Lazy cons の実現

signal により getcell が起動されると，新セルアドレスが返されるとともに，wcarentry, wcdrentry が発火し，新セルの car (cdr) 部にそれぞれゲートの入力トリが名 a, b が書込まれ，code タグがセットされる。car (cdr) 命令でセルをアクセスしたとき，code タグが付されていれば car (cdr) 部に書かれたトリが名を使って，ゲートを開ける要求シグナルを出し（これによって cons の引数評価が開始される），code タグを not ready （バリュー未到着）に変更する。car (cdr) の実行はバリューが到着するまで待たされる。

この Lazy cons を用いると無限リストの操作が可能となり，図10に示すような n 個の素数を求めるプログラムが実行できるようになる。

```
-- generating infinite integer sequence
intseqfrom: function (m) return (list)
= lazycons(m,intseqfrom(m+1));

primenumber: function (n) return (list)
= sieve(1,n,intseqfrom(2));

sieve: function (i,n,m) return (list)
= if i=n then cons(car(m),nil) else
  cons(car(m),sieve(i+1,n,delete(car(m),cdr(m))))

delete: function (x,n) return (list)
= if remainder(car(n),x)=0
  then delete(x,cdr(n))
  else lazycons(car(n),delete(x,cdr(n)))
```

図10　n個の素数を求めるプログラム

5.2　並列処理とオーバーヘッド

これまでに，関数の部分的実行，Lenient cons，色つきトークン等の種々の工夫をすることにより，データフローマシン上で並列性をうまく引出し得ることを述べてきた．しかしそれとは裏腹に並列処理を行なう故のオーバーヘッドが深刻な問題となる．

その一つが並列操作させるために必要なT/Fゲート，switchゲートなどの数の増加である．また，関数リンケージの際の"Orgating"もオーバーヘッドとなる．Orgateを使用する代りに，Lenient consの実現のときに用いた起動signalを用い，予め関数をstand byさせておけば，リンケージ時間はさらに短縮される．（しかしこの場合は条件式が

出現する度に起動 signal を switch ノードで切り分けねばならない。)

さらに有限資源である関数管理のための instance 名の管理も大きな問題である。instance 名はその instance 名を持ったトークンがなくなればくずとして解放される。しかし全てのトークンがなくなったことを確認するためには，処理が済んでトークンが送出されたという完了 signal を andgate で受けることが必要になる。

パケット通信を実現する際の通信のオーバーヘッドの問題はデータフローマシンに限らず，多数個の資源を並列に動かす場合に生じる問題である。

並列処理の陰にこれらのオーバーヘッドが隠せるか否かが，データフローマシンを現実的なものにする鍵を握っている。

5.3 履歴依存性

これは関数型言語に共通して言えることであるが，value メカニズムに基づくデータフローの原理には履歴 (storage) の概念は存在しない。しかし実際にはデータベースの更新のように storage そのものを扱わなければならない問題は多くあり，履歴依存性のある処理をどのようにデータフローマシンで実現するかが大きな問題になっている。

前の状態によって処理の内容を変更する state 概念を持っ

た計算（例えば3.2で述べたorgateなど）は，図11に示すようにown value（前の計算結果）を用いて計算を行なうループ機構により実現できる．

ループは再帰概念（tail recursion）で扱うことができるので，内部を純関数で構成し，外部には履歴依存性のある関数 $f_s(x)$ としてみせることが可能である．

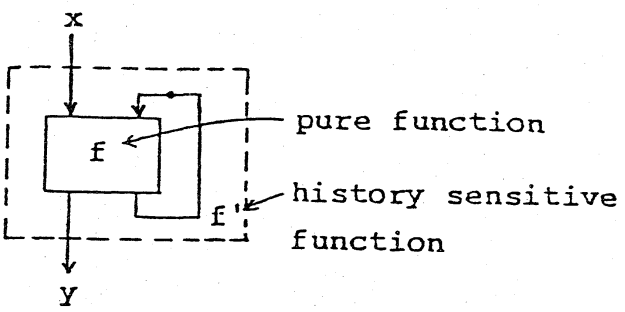


図11 履歴依存性をもつ関数

次にデータベースを多数のユーザが共有して使用する場合の資源管理の問題を考えてみよう．

データベースDの中のitem iをaに変更する計算 D' = replace(D, i, a) を純関数的に扱うと，変更される度に新たなデータベースがコピーされて作られることになる．しかしこれは現実的な解ではあり得ない．実際には資源（データベース）を共有しながら使用するので，データベースが並列にアクセスされるときの排他制御をどのようにするか，また関数性を保存して矛盾なくデータベースの中身を変えるにはどうしたらよいか，という問題が生じる．

排他制御では並列アクセスに順序づけを行なう何らかのserializerが必要であるがstream[12]の概念を導入することが有効であると思われる。

D' = replace (D, i, a) によってitemそのものを変更するとき，無矛盾性を保証するには，変更を行う前にそのitemが他で使用されていないことを確かめなければならない。これを効率よく行なう，うまい解は今の所みつかっていない。

6. おわりに

本稿では，従来型プログラムと関数型プログラムの比較を行ない，関数型言語の特質について論じた。また関数型言語とデータフローマシンが親和性に富むことを指摘し，並列性，効率を向上させる種々の機構について提案してきた。

データフローマシンには5節で述べた問題の他に，非決定性制御をどのように扱うかなど原理的な問題が一部残されている。

我々の研究室では，これらの問題を解明すると共に方式評価を行なうため，高級言語Validの作成，ソフトウェアシミュレータによるリスト処理特性の解明，circular pipeline型データフローマシン実験機と構造メモリの設計・試作を進

めている．これらの結果については稿を改めて論ずることにする．

参考文献

(1) McCarthy J., "Recursive Functions of Symbolic Expression and Their Computation by Machine", Comm. ACM, 3(4) 1960, pp. 184-195.

(2) Ackerman W.B. and Dennis J.B., "VAL - A Value Oriented Algorithmic Language Preliminary Reference Manual", MIT/LCS/TR-218, Jun 1979.

(3) Arvind, Gostelow K.P. and Plouffe W., "An Asynchronous Programming Language and Computing Machine", TR-114a, Univ. of California, Irvine, 1978.

(4) 雨宮，"データフローマシン用高級言語 Valid の設計思想"，昭56 信学会全大 NO. 1486

(5) 雨宮，長谷川，三上，"リスト処理向きデータフローマシンの検討"，情処学会記号処理研究会，13-3，1980

(6) 雨宮，長谷川，三上，"リスト処理向きデータフローマシンアーキテクチャとそのソフトウェアシミュレータ"，信学会電子計算機研究会，技法 EC-80-69，1981.

(7) Friedman D.P. and Wise W.S., "Cons should not evaluate its arguments", Automata, Language and Programming, Edinburgh Univ. Press, 1976.

(8) 雨宮, 長谷川, "リスト処理向きデータフローマシンの構造体メモリと Lenient cons の実現について", 情処学会第22回(昭56前)全国大会 4J-9.

(9) 長谷川, 雨宮, "データ駆動制御下での並列アルゴリズムの評価," 情処学会第23回(昭56後)全国大会. 4H-3

(10) 中村, 長谷川, 雨宮, "リスト処理向きデータフローマシン用構造体メモリの設計と評価", 信学会電子計算機研究会, 技法 EC81-32, 1981.

(11) Dennis, J.B., "The Varieties of Data Flow Computers", Proc 1st Int. Conf. on Distributed Computing Systems, 1979, pp. 430-439.

(12) Arvind, "A Multiple Processor Data Flow Machine that Supports Generalized Procedures, 8th Annual Symp. on Computer Architecture, 1981, pp. 291-302.